

CPEN 311: Digital Systems Design

Asynchronous State Machines

Introduction

Most real digital systems are purely synchronous, that is, they operate on a single clock. These are easy to design, but have some disadvantages. It is also possible to design a circuit that does not depend on a clock. In this slide set, we will talk about such circuits. Even though you may not see a huge system that is entirely asynchronous any time in the near future, you will likely come across small asynchronous circuits, so it is important that you have seen them before.



Problems with Synchronous Design:

1. In Synchronous design, clock period is dictated by the longest path
“in the MIPS R10000, there was a single long path in the processor’s instruction fetch hardware. This long path was limiting the achievable clock frequency, but the engineers couldn’t find it! They finally found it and shortened it for the MIPS R12000”.
2. Since everything happens on the clock edge, instantaneous power is a problem
3. Excessive noise at the frequency of the clock
4. In large chips, distributing the clock is difficult

In Practice Though...

- Most real digital systems are purely synchronous.
- Synchronous systems are (relatively) easier to design due to simpler timing considerations

Why learn about asynchronous design?

- You will almost never see large entirely asynchronous systems, but you will come across small asynchronous circuits.
- Understanding asynchronous circuits will give you a deeper understanding of synchronous circuits
- Some applications are naturally more suited to an asynchronous design (e.g. those that react to asynchronous input)

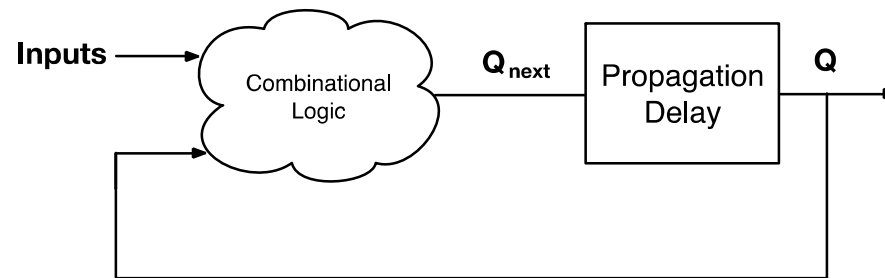
Learning Objectives

1. Be able to design functional asynchronous finite state machines
2. Understand why asynchronous FSMs are susceptible to glitches
3. Understand how to avoid glitches
4. Understand why state encoding is important for asynchronous FSMs
5. Understand how to make suitable state encoding assignments

In the “real world”, it will be important that you can design simple asynchronous state machines. It is unlikely you will ever be designing anything really big using asynchronous techniques

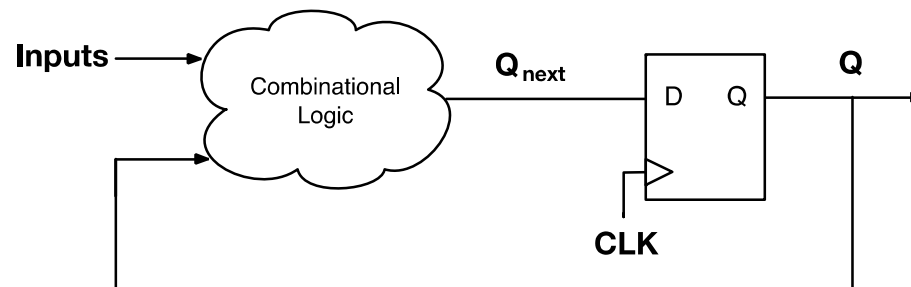
Asynchronous Finite State Machines

Asynchronous FSM



Transitions state **AS SOON AS POSSIBLE**

Synchronous FSM



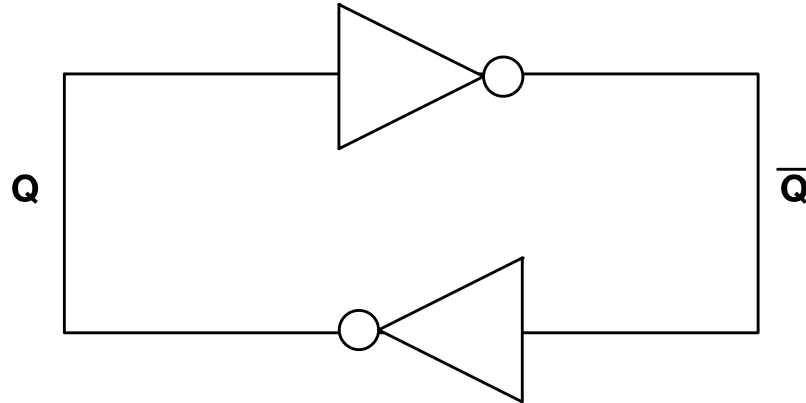
Transitions state **ON CLOCK EDGE**

Note: Output logic not shown. Next state logic shown only

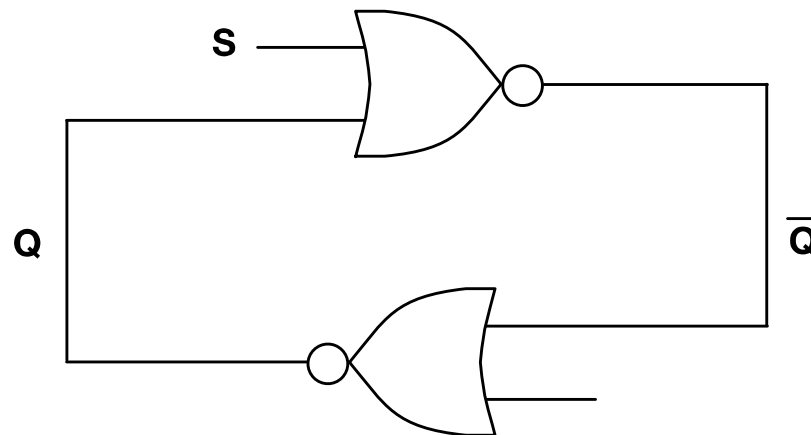
Latches and Flip-Flops are Asynchronous Circuits On the Inside!

Recall the Fundamental Storage Element

Makes use of a **COMBINATIONAL LOOP** for storage

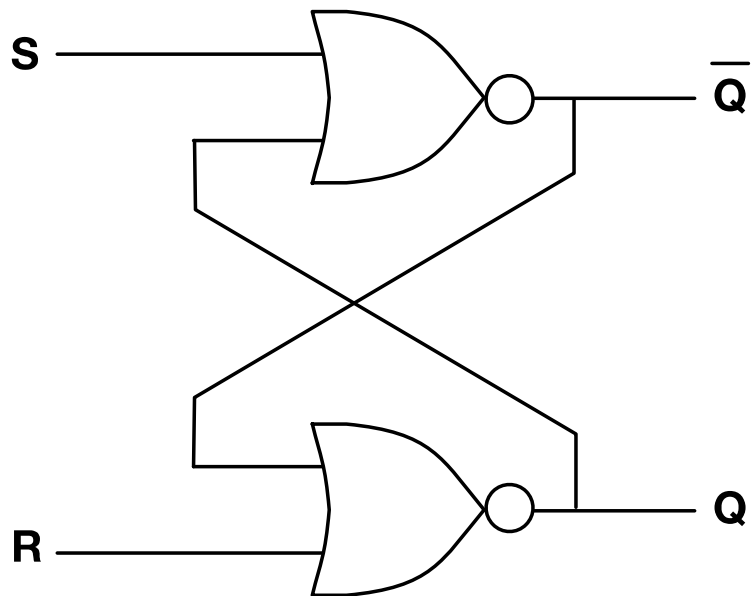


One way to “write” to this element



This is an SR Latch!

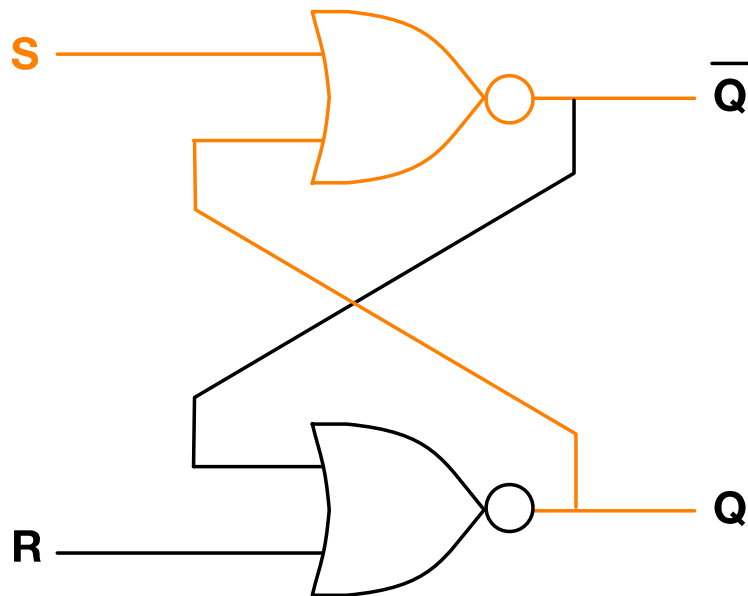
Review – SR Latch



S	R	Q
0	0	Hold Value
0	1	0
1	0	1
1	1	Not Used

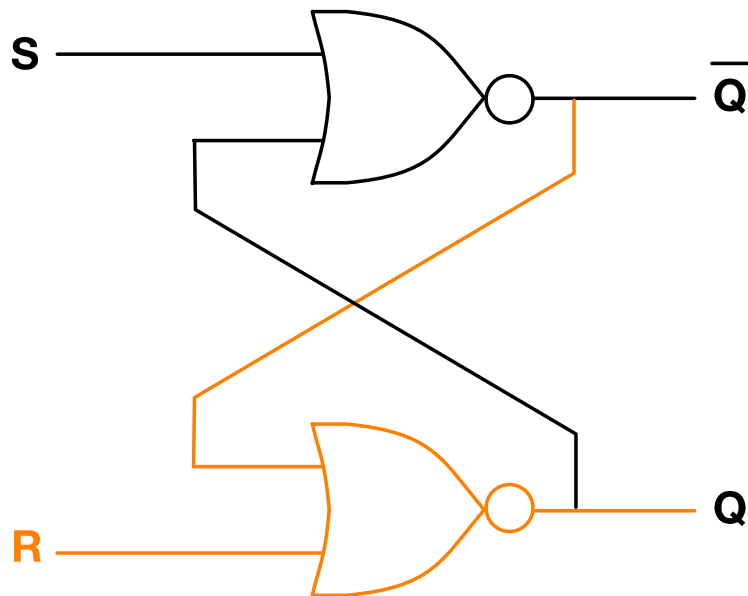
THIS IS A STATE MACHINE!

SR Latch – Set (Make Output = 1)



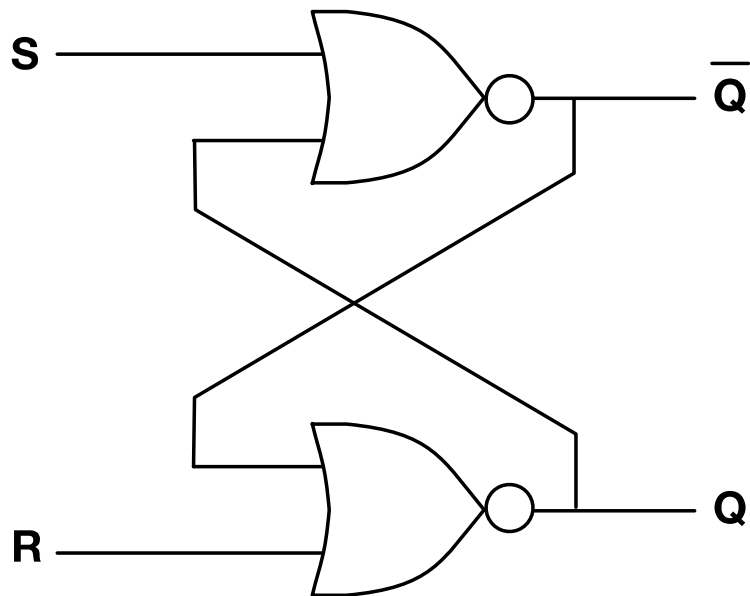
S	Q	$\overline{Q}_{\text{next}}$	
0	0	1	} Q keeps old value
0	1	0	
1	0	0	} \overline{Q} forced to 0 (Q forced to 1)
1	1	0	


SR Latch – Reset (Make Output = 0)



R	\overline{Q}	Q_{next}	
0	0	1	} Q keeps old value
0	1	0	
1	0	0	} Q forced to 0 (\overline{Q} forced to 1)
1	1	0	

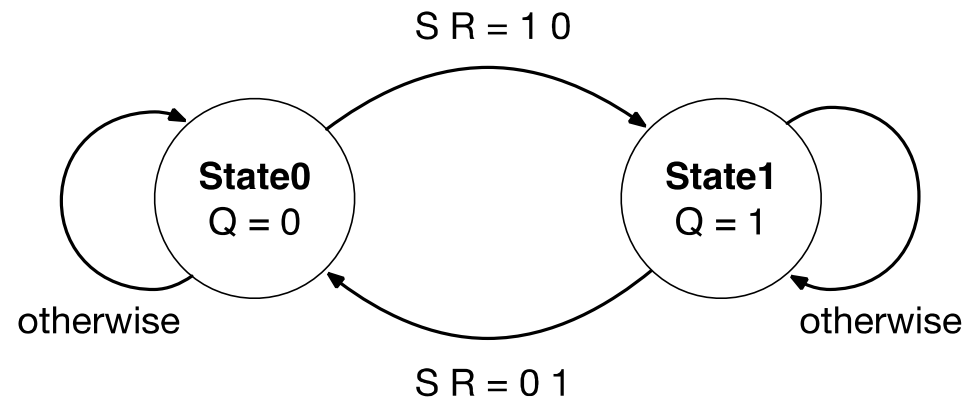
SR Latch – State Transition Table



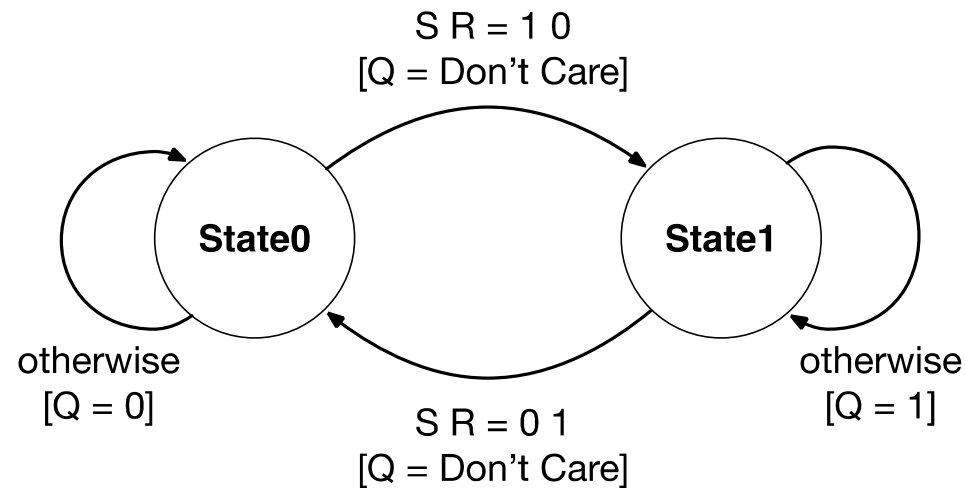
S	R	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	

State Diagram For SR Latch

Moore Representation

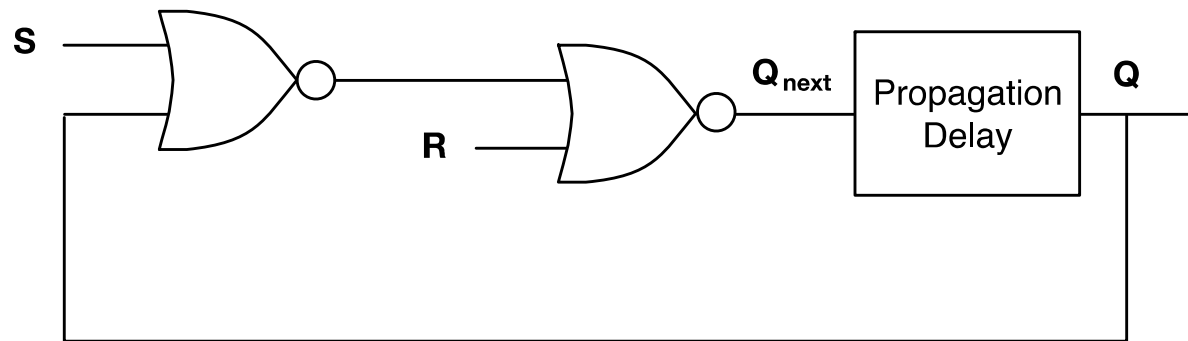


Mealy Representation

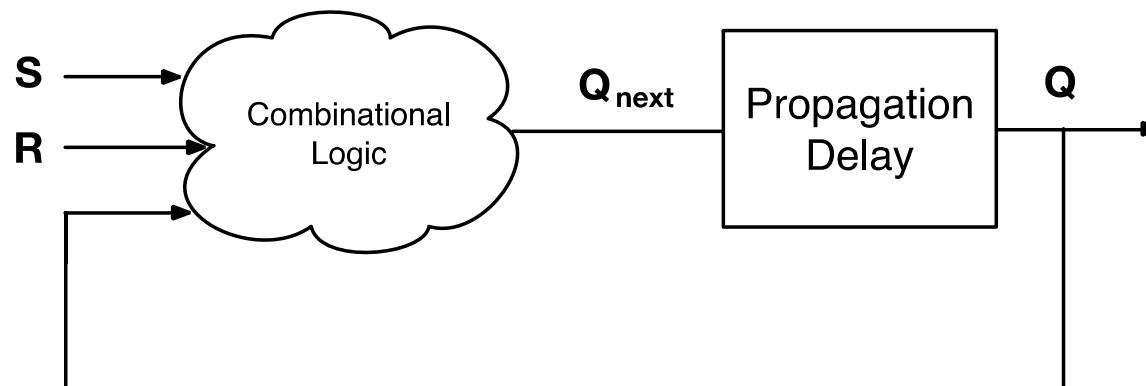


When does $Q \leftarrow Q_{\text{next}}$?

State changes instantly once inputs change
(almost ... it changes after some propagation delay)

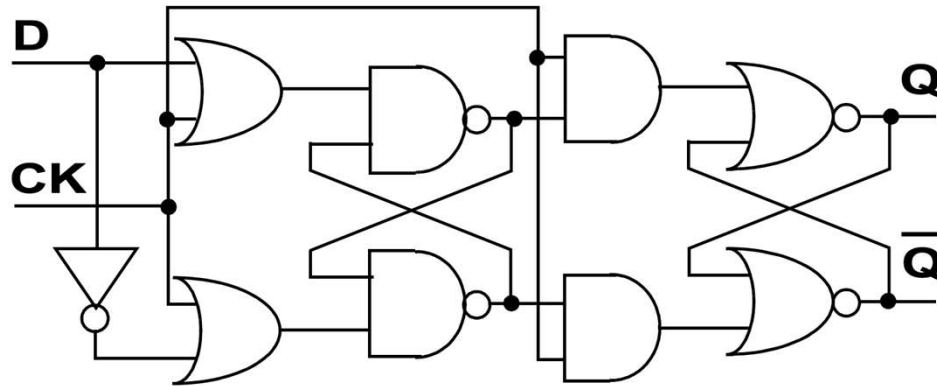


Note: This figure is an abstraction. Delay is distributed across gates.

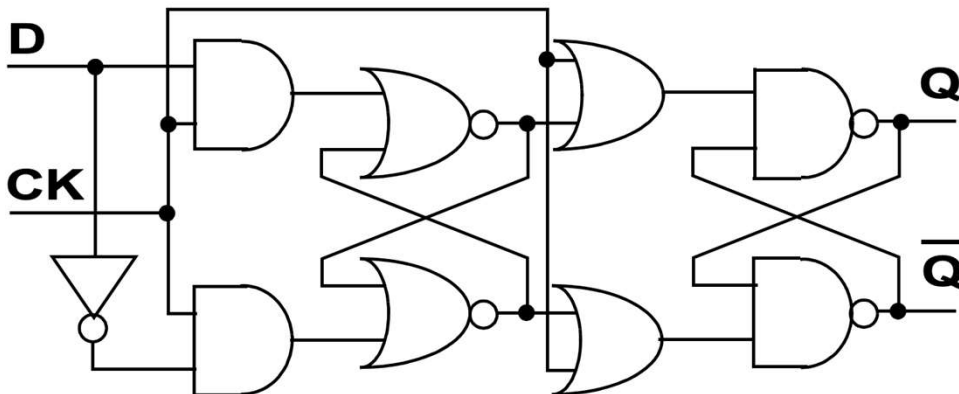


No Flip-Flops and no clock!

DFFs



posedge-triggered DFF:
active-low latch,
then active-high latch



negedge-triggered DFF:
active-high latch,
then active-low latch

Designing Asynchronous FSMs

Design like you would a Synchronous FSM (CPEN 211)

1. Start with a State Diagram
2. Assign Encoding to States
3. Create State Transition Table
4. Create Karnaugh Map from State Transition Table
5. Write boolean equations for next-state

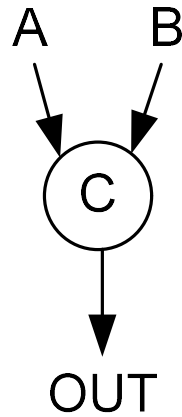
Two Concerns for Async FSM Design:

1. Glitches
2. State Encoding
3. Immediate Transitions

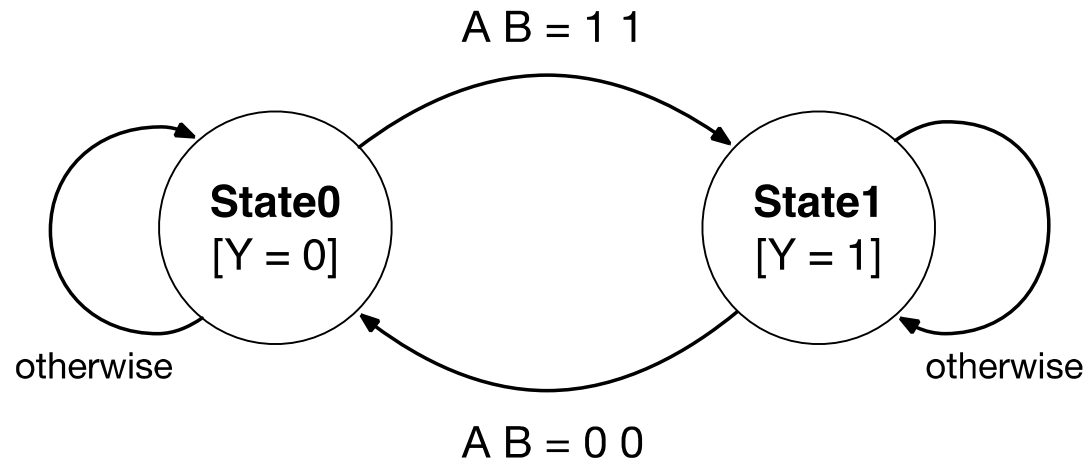
Best way to learn is through examples ...

Design Example: Muller C Element

Design Example – Muller C Element

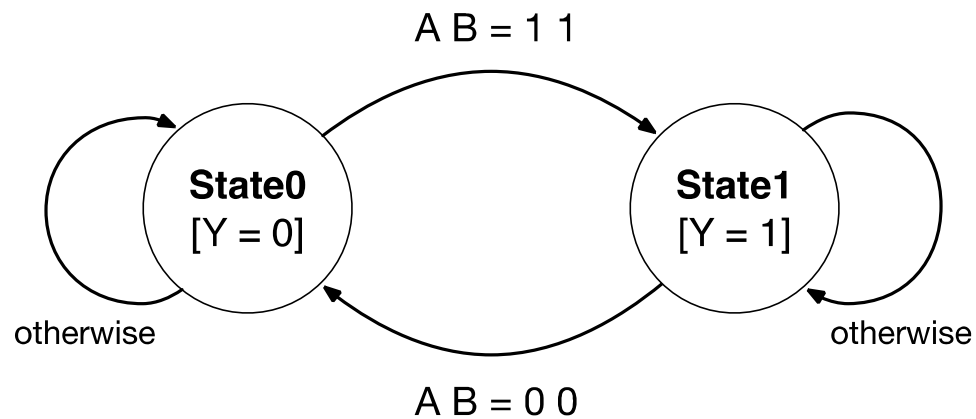


A	B	Y
0	0	0
0	1	Unchanged
1	0	Unchanged
1	1	1



State Assignment

Only two states so only need one state variable



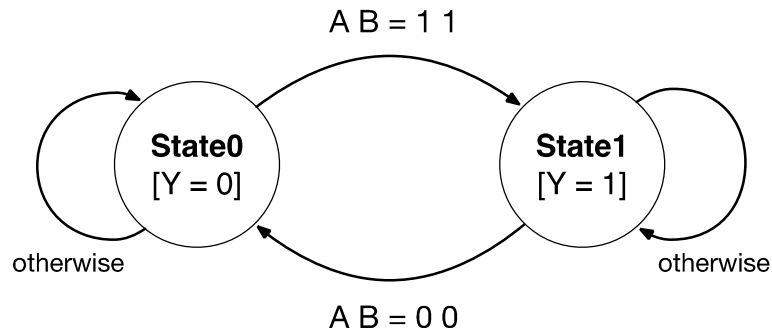
State Variable: Q

**State Encoding
Assignment:**

State0 → 0

State1 → 1

State Transition Table



State Variable: Q

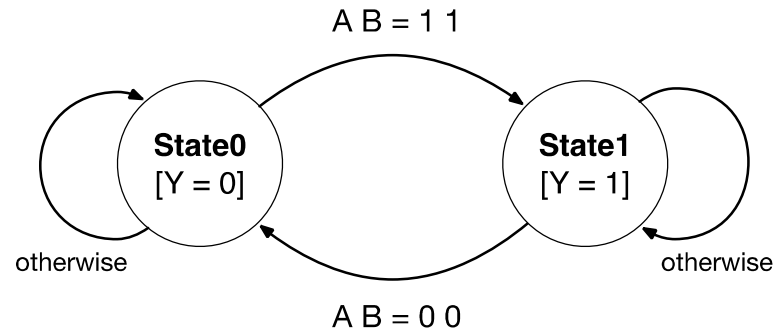
State Encoding Assignment:

State0 → 0

State1 → 1

Present State	Next State			
	AB = 00	01	10	11
0	0	0	0	1
1	0	1	1	1

Create K-Map From State Transition Table

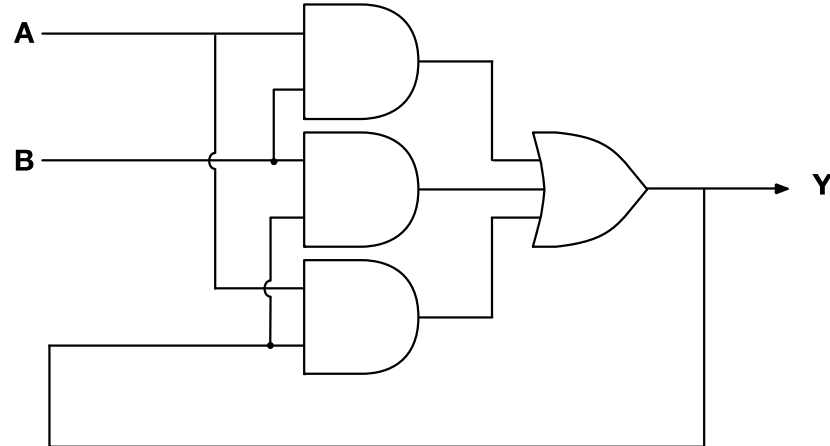
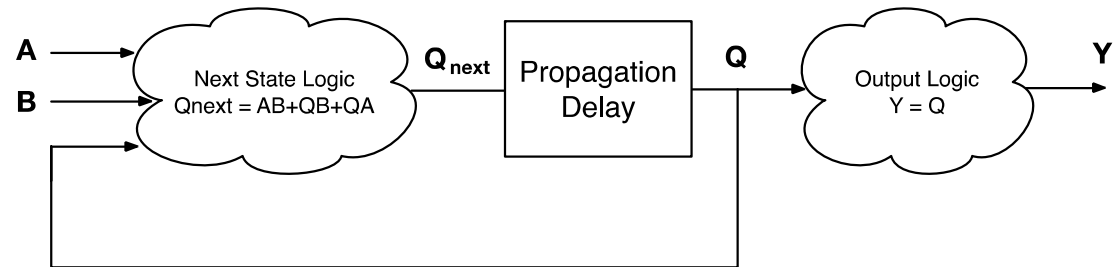


Q_{next}		$AB =$			
		00	01	11	10
Q	0	0	0	1	0
	1	0	1	1	1

$$Q_{next} = AB + QB + QA$$

Final Circuit

$$Q_{\text{next}} = AB + QB + QA$$



No Flip-Flops!

Fundamental Difference

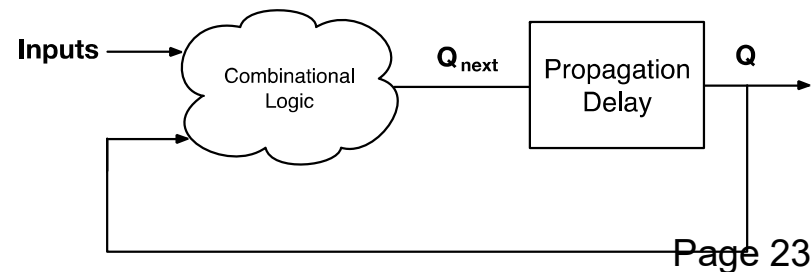
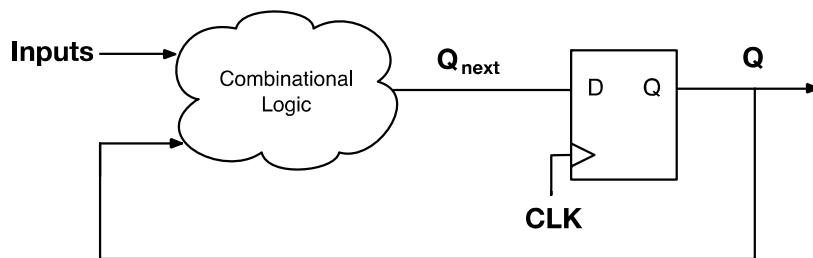
When next state is not the same as current state...

- Synchronous FSM waits for next rising clock edge to change states
 - everything is **SYNCHRONIZED** to the clock
- Asynchronous FSM **IMMEDIATELY** changes state
 - need to synchronize in other ways if necessary

Another way to think about it:

An Asynchronous FSM has an imaginary clock, whose period is exactly the same as the propagation delay through the combinational feedback path.

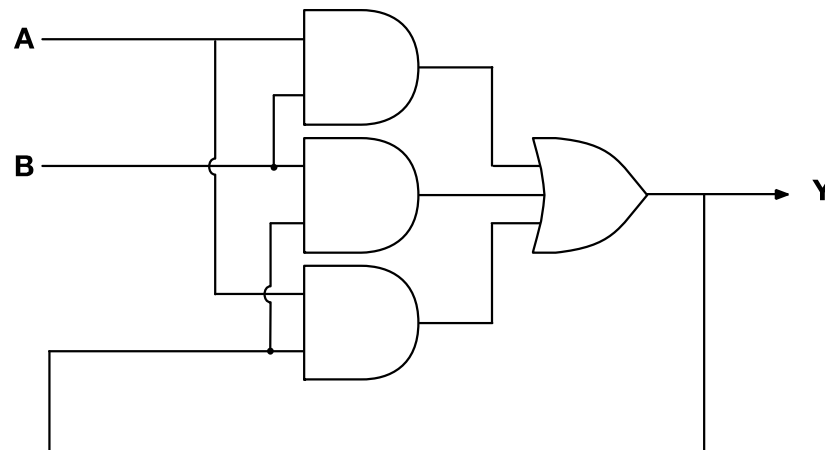
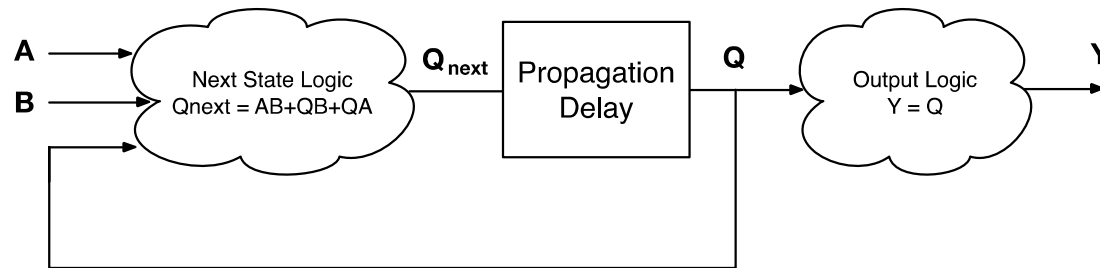
Performance: Worst-Path Delay vs. Average-Path Delay



Stability

Stability

The circuit is stable when, for a given set of next-state logic input values (including feedbacks), the circuit reaches and remains in a particular state.



In the previous example, we can see that the machine is:

Stable when either of:

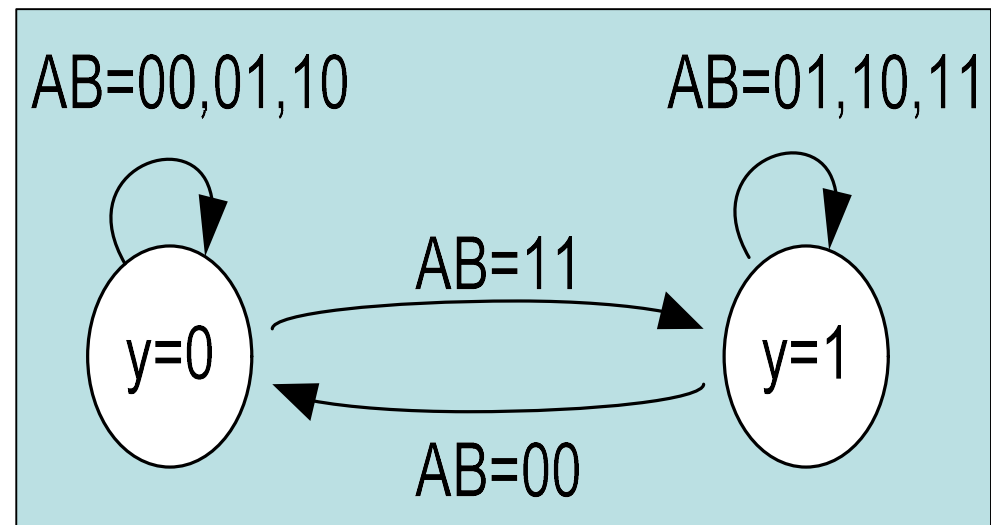
In state S1 and AB=00, 01, or 10

In state S2 and AB=01, 10, or 11

Unstable when either of:

In state S1 and AB=11

In state S2 and AB=00



Why is this?

When we are in state S1, and we get a 11, we immediately go to S2.
Once we reach S2, we are stable again.

When we are in state S2, and we get a 00, we immediately go to S1.
Once we reach S1, we are stable again.

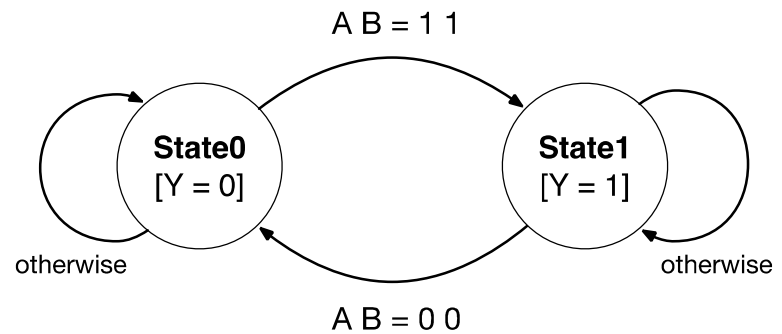
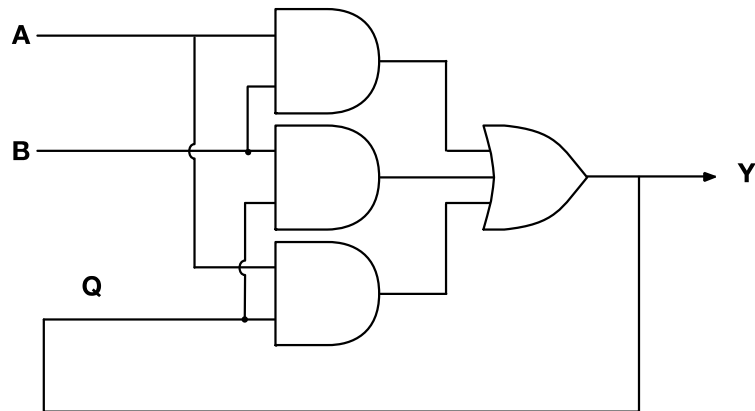
This is a fundamental difference between a synchronous and asynchronous state machine:

In a synchronous machine, when the “next state” is not the same as the “current state”, we wait until the next rising clock edge to actually change state.

In an asynchronous machine, when the “next state” is not the same as the “current state”, we immediately make a transition to the next state.

- The next state may also not be stable, in that case, we would calculate a new next state, and immediately make a transition there, and so on....
- To be useful, we have to eventually reach a stable state.

Stability



When $Q_{\text{next}} \neq Q$ circuit is **UNSTABLE**

Q_{next}	$AB = 00$	01	11	10
Q 0	0	0	1	0
1	0	1	1	1

Immediately transitions
from State1 \rightarrow State0

Immediately transitions
from State0 \rightarrow State1

Glitches

Glitches

In a synchronous system,

- Glitches are ok if they resolve before the next rising clock edge.

In an asynchronous system,

- Every signal transition has meaning
→ Glitches may cause system to transition into unexpected state

RULE: Next-State and output logic must be glitch-free

Terminology

Glitch

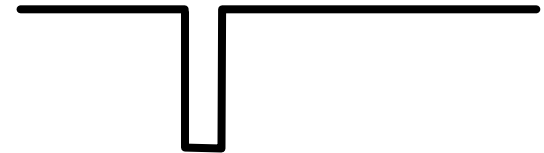
A temporary unwanted pulse/spike in a signal

Hazard

A possibility of glitching due to circuit structure. In other words, a circuit with the potential to glitch is said to have a hazard.

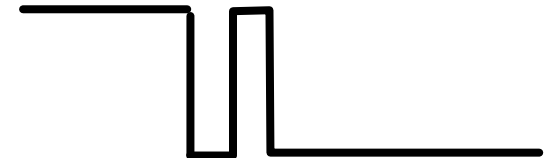
Static Hazard

The potential for glitching to occur when the signal value should not change (remain static)
Transitions when there shouldn't be any.

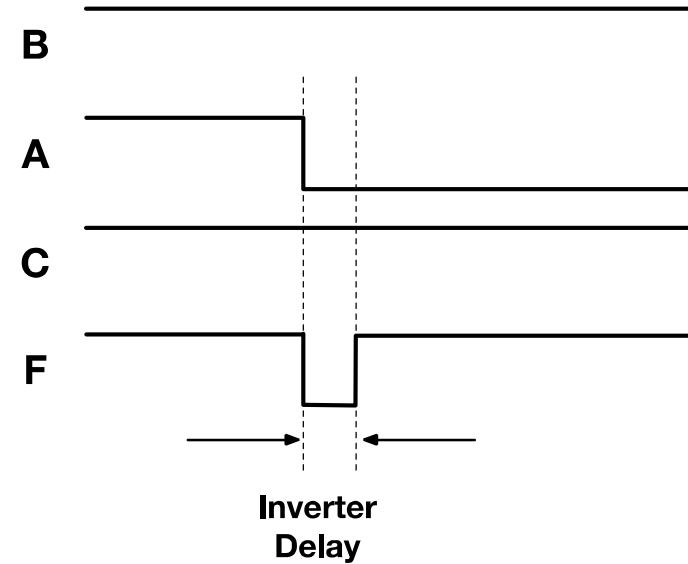
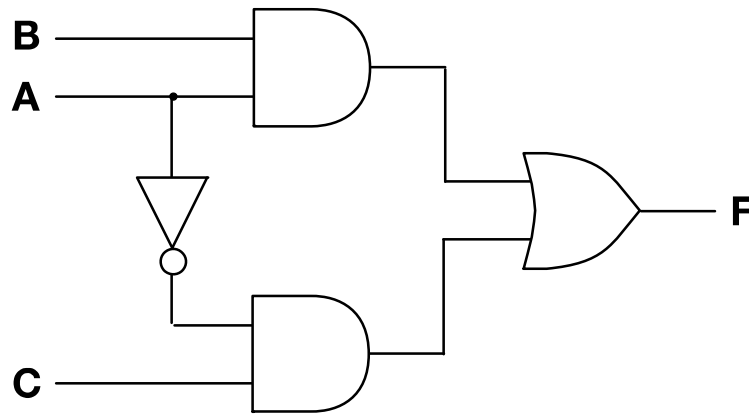


Dynamic Hazard

The potential for glitching to occur when the signal value should change (dynamic).
Multiple transitions when there should be one.



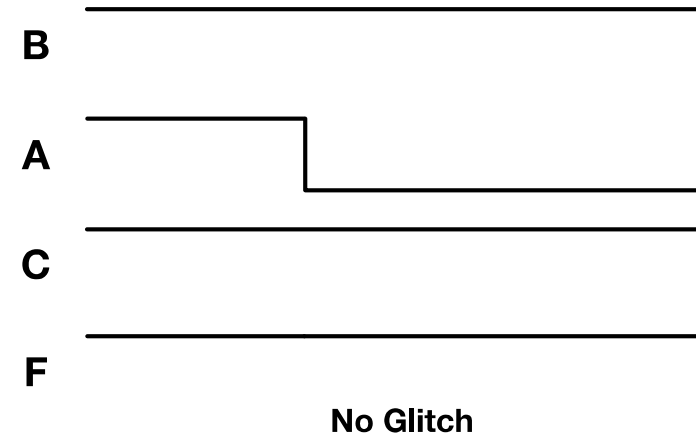
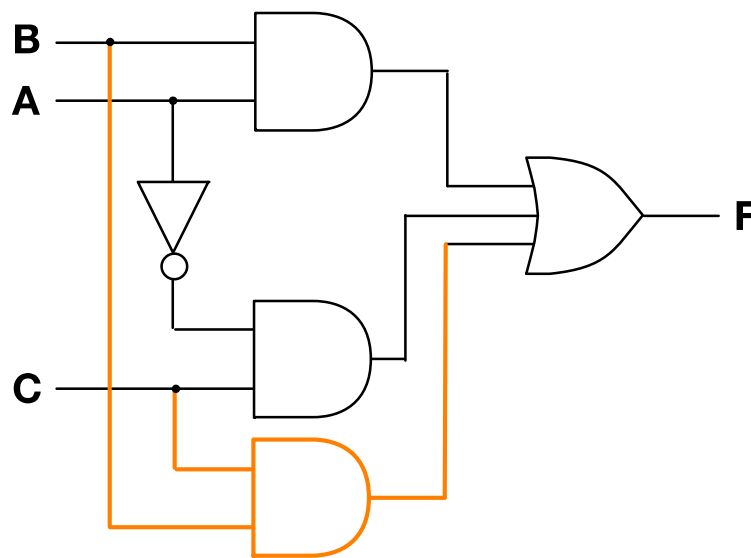
Static Hazard Example



F		AB =			
		00	01	11	10
C	0	0	0	1	0
	1	1	1	1	0

Adjacent Non-Overlapping Covers can cause glitches

Eliminate Static Hazard – Mask Glitch



F		AB =			
		00	01	11	10
C	0	0	0	1	0
	1	1	1	1	0

Add overlapping cover to **MASK** glitch

Glitches were never a problem for synchronous circuits; as long as they stabilized by the next rising clock edge, we were fine. Here, if the output goes low even for a short time, this may cause us to go into an unexpected state.

RULE: Make sure your next state logic and output logic is glitch-free!

State Encoding Assignment

State Encoding Assignment

RULE: Hamming Distance of encoding between adjacent states must be 1

OK

000 → 010

Not OK

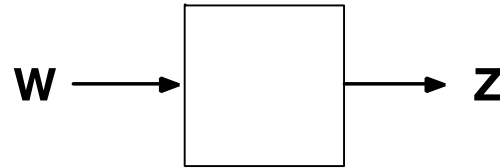
000 → 011

Only one state variable can change at a time

Why?

Let's look at another design example ...

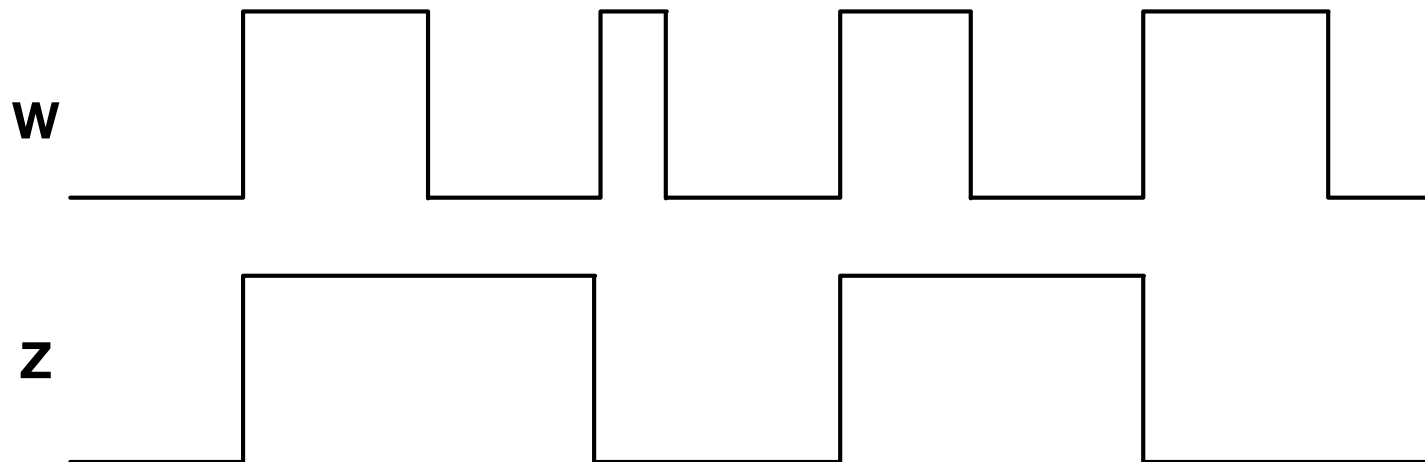
Pulse Counter



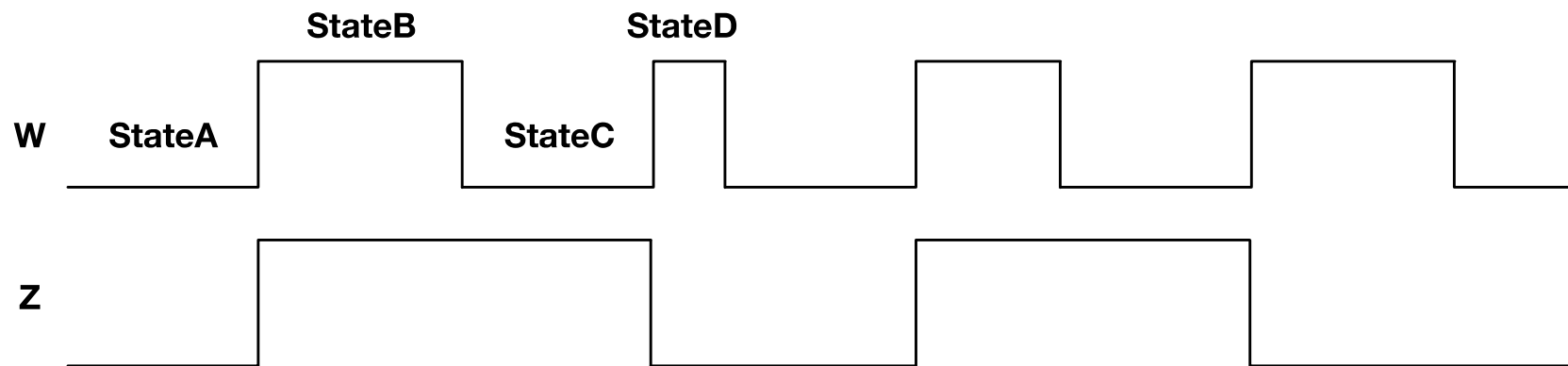
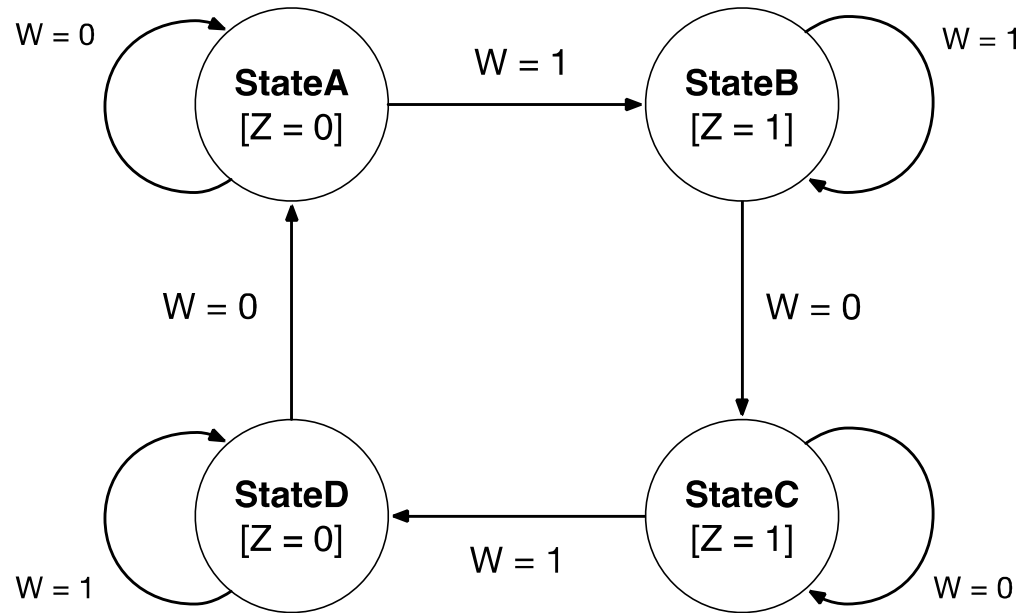
One input **W**, one output **Z**

When pulse appears on **W**

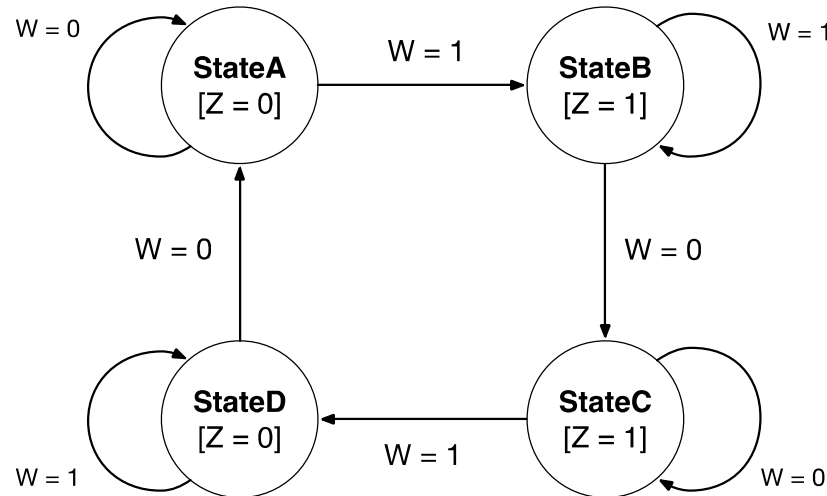
- **Z = 0** if the number of previously applied pulses is **even**
- **Z = 1** if the number of previously applied pulses is **odd**



1. Create State Diagram



2. Assign State Encoding



State Variables: $Y_1 Y_0$

StateA \rightarrow 00

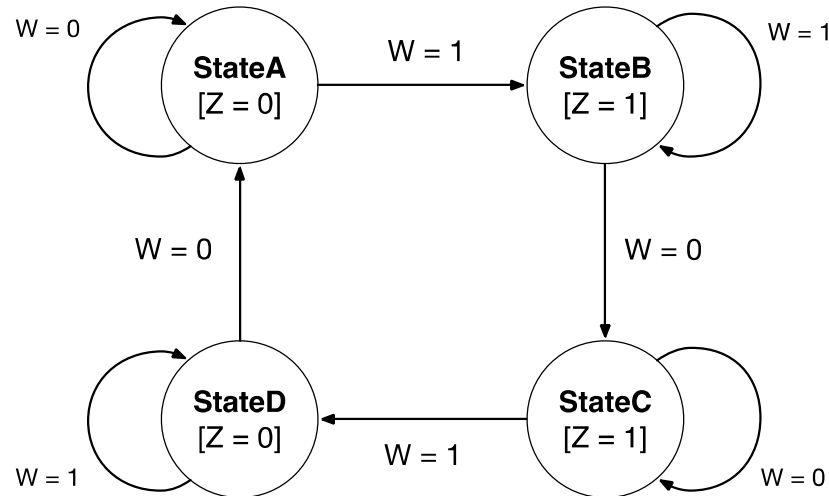
StateB \rightarrow 01

StateC \rightarrow 11

StateD \rightarrow 10

Note: Hamming Distance = 1 for adjacent States

3. Create State Transition Table



State Variables: Y_1Y_0

StateA \rightarrow 00

StateB \rightarrow 01

StateC \rightarrow 11

StateD \rightarrow 10

Present State $Y_1 Y_0$	Next State		Output Z
	$W = 0$	$W = 1$	
	$Y_1 Y_0 \text{ next}$		
00	00	01	0
01	11	01	1
10	00	10	0
11	11	10	1

4. Construct K-Map and Write Equations

Construct Karnaugh Map for each state and output variable.

		$Y_1 Y_0$			
$Y_{1 \text{ next}}$		00	01	11	10
w	0	0	1	1	0
	1	0	0	1	1

$$Y_{1 \text{ next}} = Y_1 Y_0 + Y_0 \overline{W} + Y_1 W$$

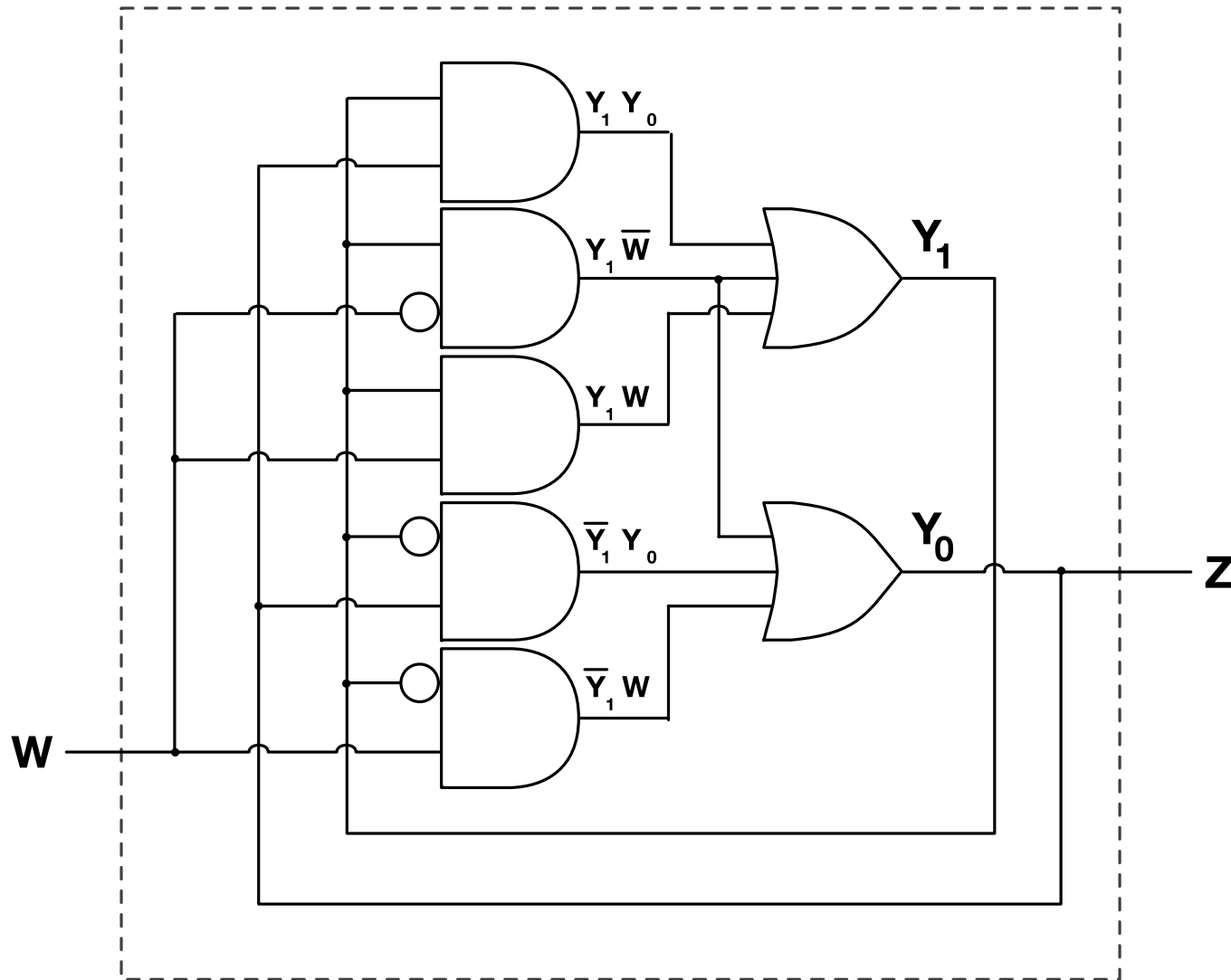
		$Y_1 Y_0$			
$Y_{0 \text{ next}}$		00	01	11	10
w	0	0	1	1	0
	1	1	1	0	0

$$Y_{0 \text{ next}} = \overline{Y}_1 Y_0 + Y_0 \overline{W} + \overline{Y}_1 W$$

By inspection, (use KMap if you can't see this):

$$Z = Y_0$$

Final Async FSM Circuit



State Assignment Revisited

We used the following state assignment and it works

State Variables: Y_1Y_0

StateA \rightarrow 00

StateB \rightarrow 01

StateC \rightarrow 11

StateD \rightarrow 10

What about the following assignment instead?

State Variables: Y_1Y_0

StateA \rightarrow 00

StateB \rightarrow 01

StateC \rightarrow 10

StateD \rightarrow 11

Different
assignment

This would lead to incorrect functionality. Why?!?

Problem When Hamming Distance > 1

Problem in StateD when $W \rightarrow 0$

BAD State Assignment

Present State $Y_1 Y_0$	Next State $Y_1 Y_0 \text{ next}$	
	$W = 0$	$W = 1$
00	00	01
01	10	01
10	10	11
11	00	11

$Y_1 Y_0$ needs to change from 11 \rightarrow 00
i.e. both signals need to change
AT THE EXACT SAME TIME
for correct operation

Synchronous FSM

- OK – both state variables change on clock edge

Asynchronous FSM

- Each state variable updates independently based on propagation delay of its own combinational loop

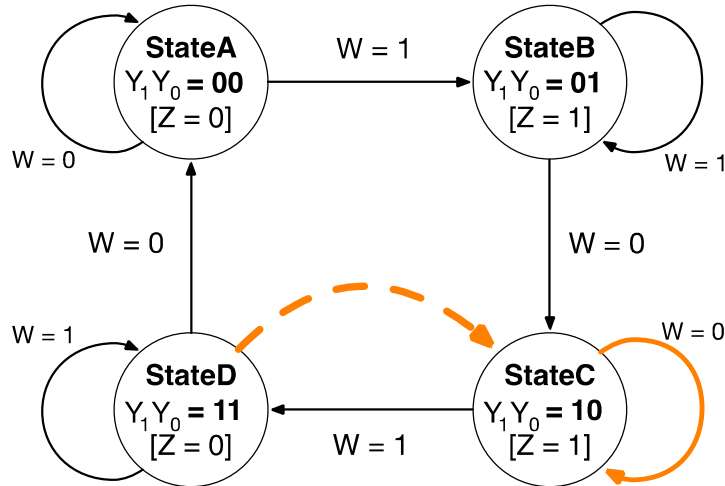
**Cannot guarantee equal delays on
all state variable feedback paths**

Most Likely Outcome

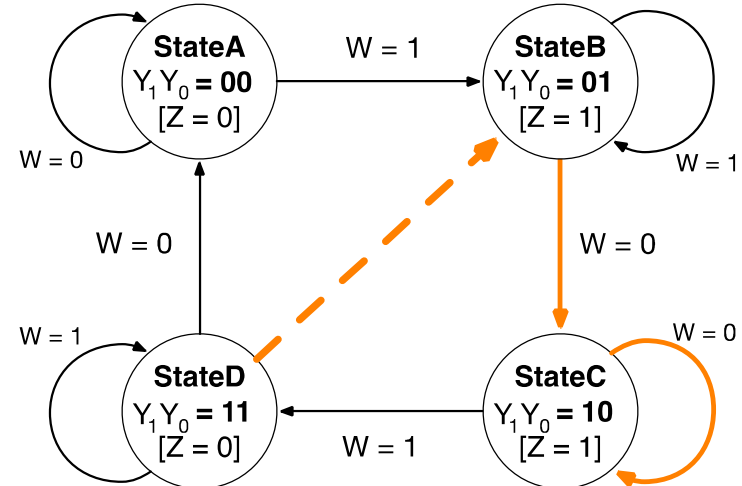
One of the state variables will change before the other

Y_0 changes to 0 first $\rightarrow Y_1Y_0 = 10$

Y_1 changes to 0 first $\rightarrow Y_1Y_0 = 01$



Since $W = 0$, FSM Stays in 10
WRONG STATE



Since $W = 0$, immediately go to 10
then stays in 10
WRONG STATE

State Assignments

RULE: Hamming Distance of encoding between adjacent states must be 1

If you can't find an appropriate state assignment to satisfy this rule, you can add extra states

Synthesizable?

Some tools have problems synthesizing combinational loops.

In synchronous systems, combinational loops are generally design errors (like inferred latches)

Altera Quartus II can synthesize them

But you will get critical warnings since in typical synchronous systems, combinational loops are unintended (like inferred latches)

Warning: Timing Analysis is analyzing one or more combinational loops as latches

What is the path delay through a loop?!?

Learning Objectives

1. Be able to design functional asynchronous finite state machines
2. Understand why asynchronous FSMs are susceptible to glitches
3. Understand how to avoid glitches
4. Understand why state encoding is important for asynchronous FSMs
5. Understand how to make suitable state encoding assignments

In the “real world”, it will be important that you can design simple asynchronous state machines. It is unlikely you will ever be designing anything really big using asynchronous techniques